Convolutional neural network

A convolutional neural network is just a network with some convolutional layer. This is useful in many cases, especially when the input data have a special structure (audio, images, videos, ...). Convolutional neural networks allows to keep the spacial structure of the input so that we don't loose informations with feature vectors.

Convolution operator for continuous functions

$$(x * w)(t) = \int_{-\infty}^{\infty} x(a)\, w(t-a)\, da$$

you multiply a function by a shifted version of another function

Convolution operator for discrete functions

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\, w(t-a)$$

1D discrete signals

Convolution operator for discrete, limited functions

$$(x * w)(t) = \sum_{m \in S_1} x(m)\, w(t-m)$$

w: kernel (1D)

Convolution operator for discrete, limited 2D functions

$$(I * K)(i,j) = \sum_{m \in S_1} \sum_{n \in S_2} I(m,n)\, K(i-m, j-n)$$

I: input; K: kernel (2D)

Convolution operator for discrete, limited 3D functions

$$(I * K)(i,j,k) = \sum_{m \in S_1} \sum_{n \in S_2} \sum_{u \in S_2} I(m,n,u)\, K(i-m, j-n, k-u)$$

I: input; K: kernel (3D)

Properties :

① Convolution is commutative

$$(I * K)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

this way we can shift the kernel over the image

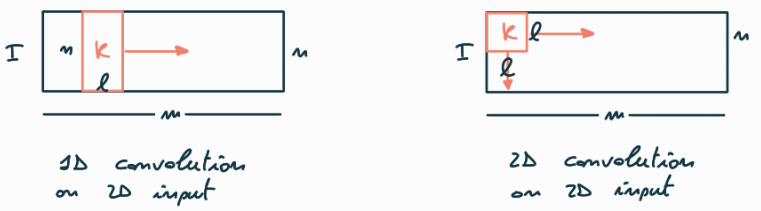
② Flipping (↕) the kernel results in a new operation : cross-correlation

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

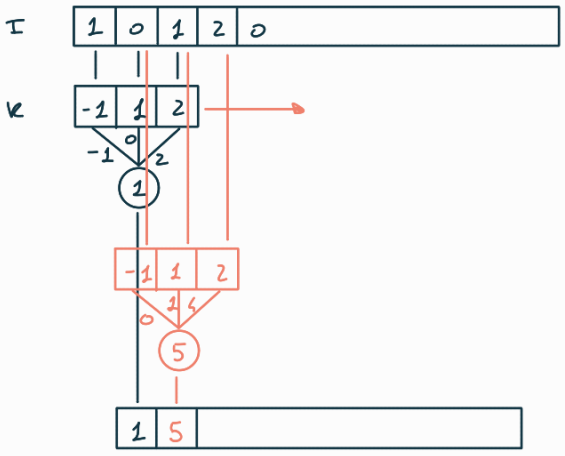
The main goal of a convolutional neural network is to learn the kernel k.

- 1D convolution → Kernel moves in one direction → 1D output
- 2D convolution → Kernel moves in two directions → 2D output
- 3D convolution → Kernel moves in three directions → 3D output

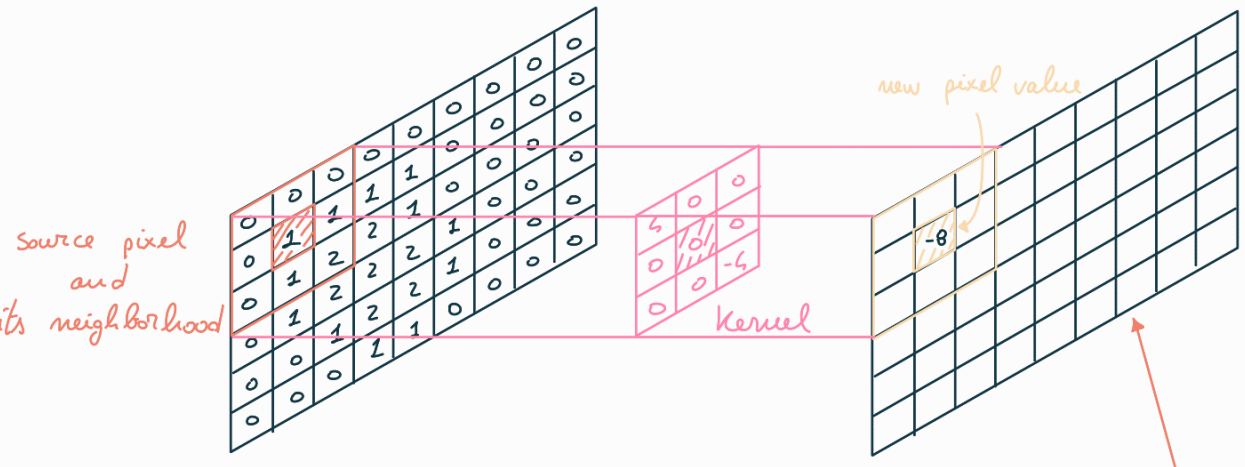
1D/2D/3D convolutions does not refer to the dimension of the input and/or kernel
The number of dimension of the output depends on the convolution type



1D convolution with 1D input and kernel



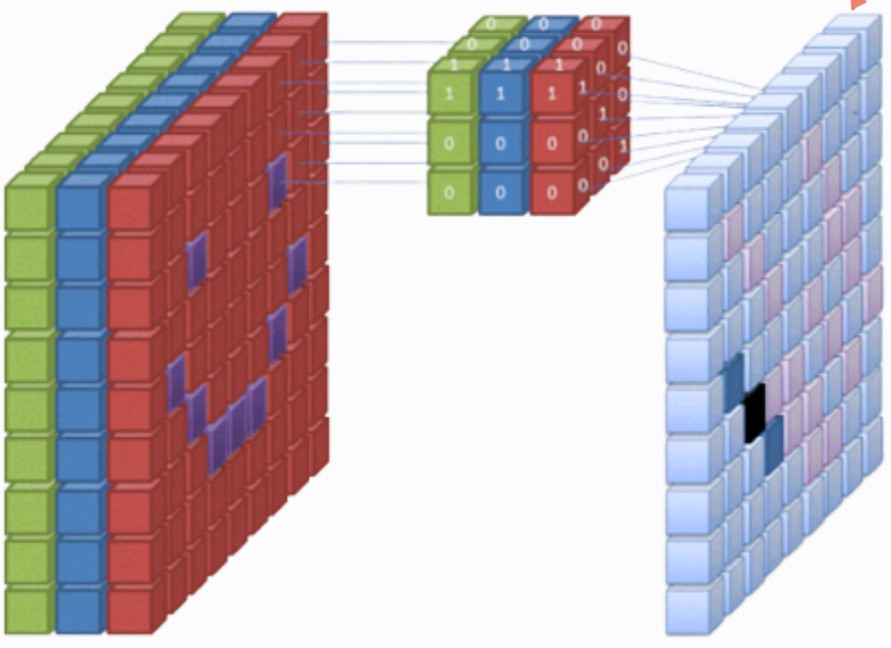
2D convolution for 2D input signal (grey scale image) with 2D kernel



(4·0)+
(0·0)+
(0·0)+
(0·0)+
(0·1)+
(0·1)+
(0·0)+
(0·1)+
(-4·2)=
-8

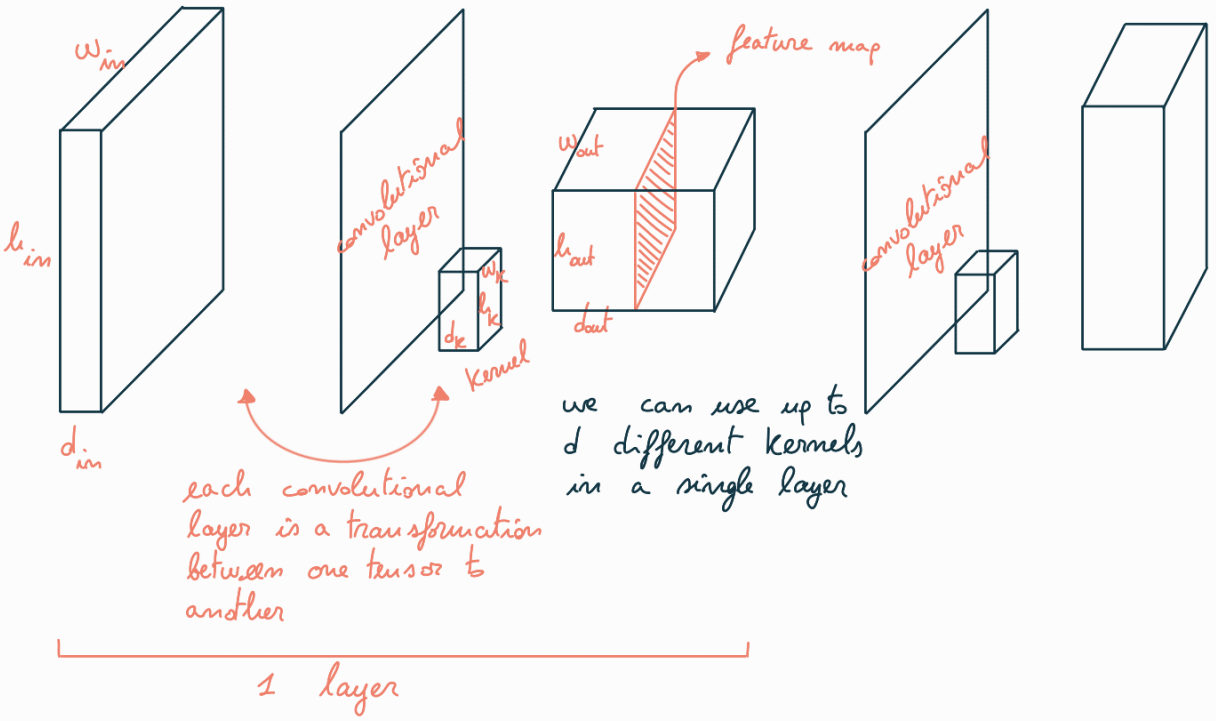
note that in both cases we have a 2D convolution and a 2D output even if the kernel size changes

2D convolution for 3D input image with a 3D kernel



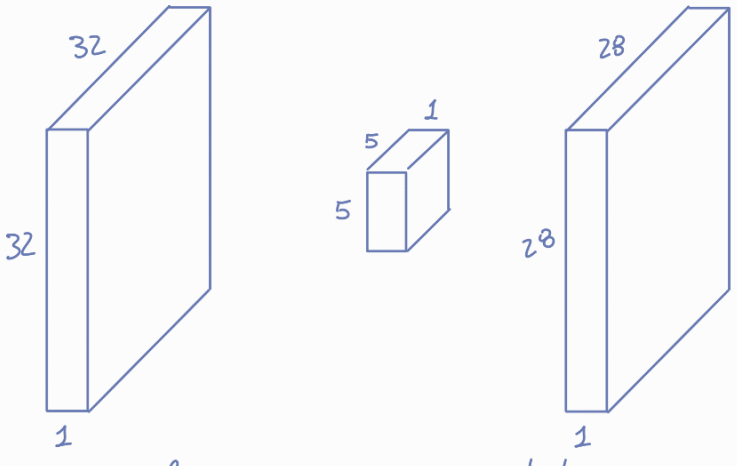
In order to obtain a 2D convolution, the depth of the kernel must be equal to the depth of the input

General structure of a convolutional neural network

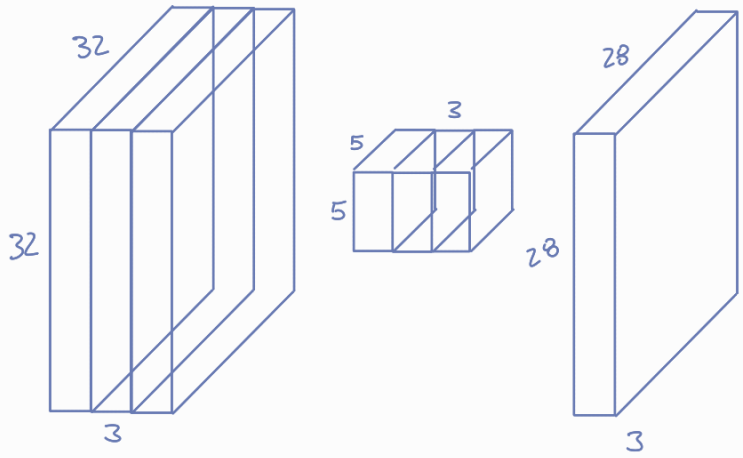


The only trainable parameters of the model are just the parameters of the Kernels

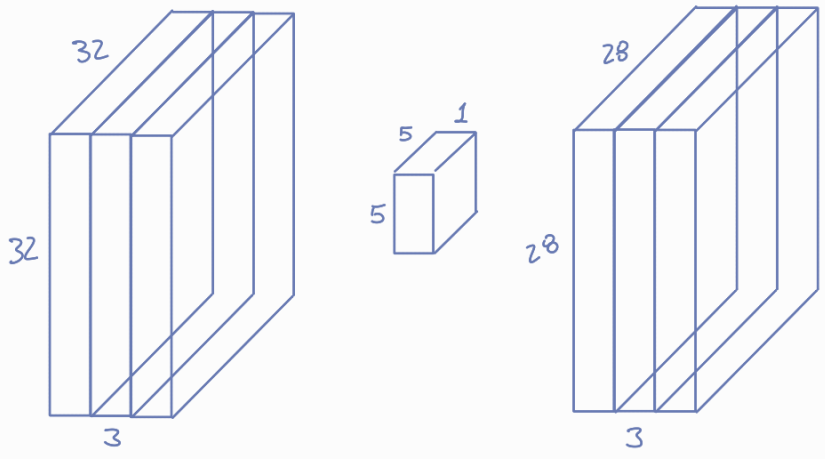
Example :



in this case we have a 2D convolution; the size of the output (aside from being 2D) is influenced by the padding and the striding

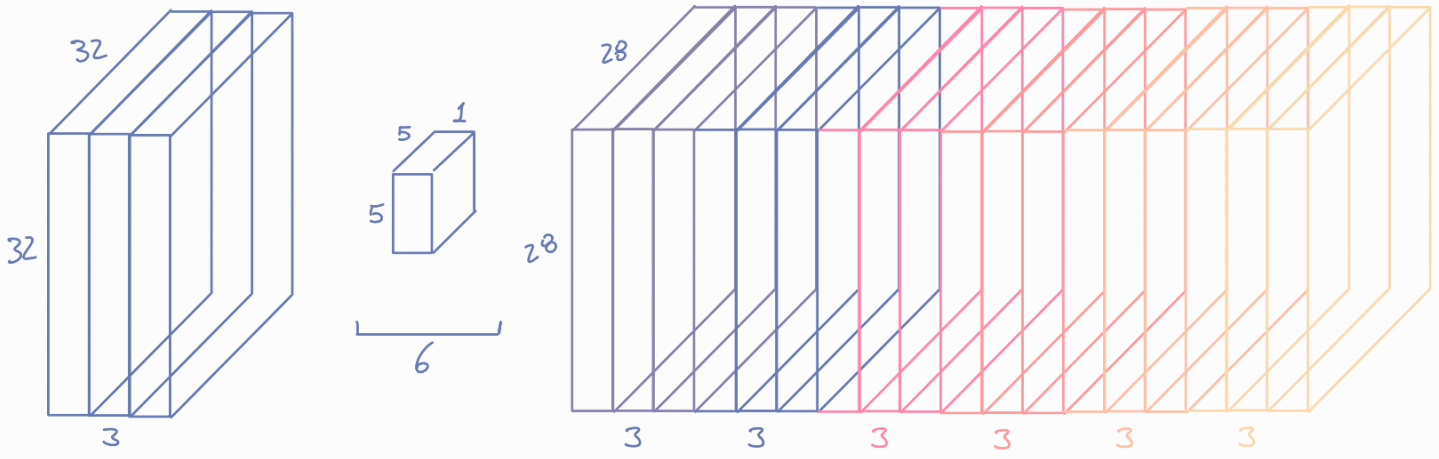


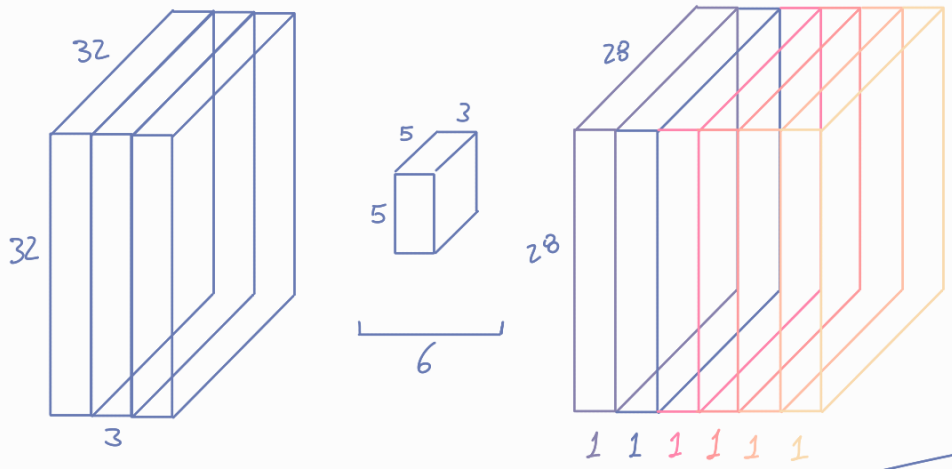
also in this case we have a 2D convolution



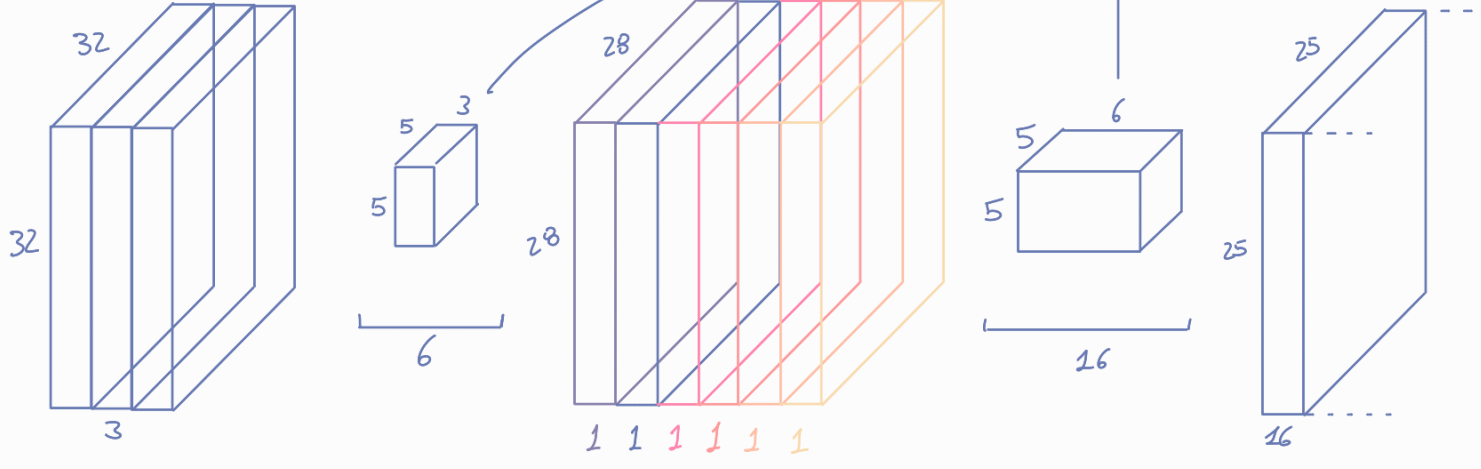
in this case we have a 3D convolution, thus a 3D output

If we use 6 kernels, we iterate this procedure (single kernel) 6 times, obtaining 6 feature maps one after the other.





Multiple layer example



In general

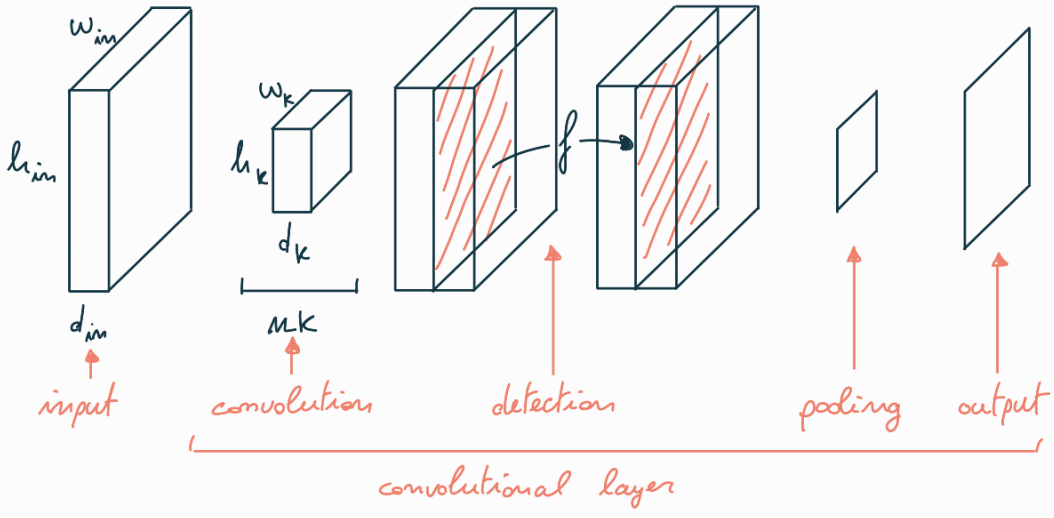
Input : $w_{in} \times h_{in} \times d_{in}$
Kernel: $w_k \times h_k \times d_k$
} if we want 2D convolutions we need to add the constraint $d_{in} = d_k$

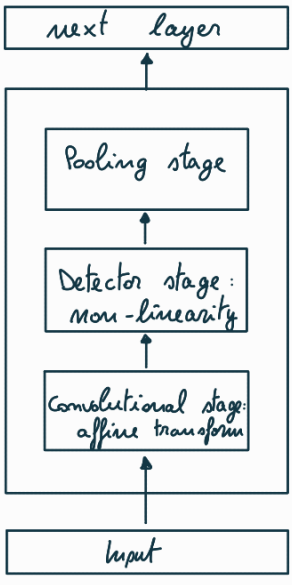
Number of parameters on the level is $w_k \times h_k \times d_k \times m\_k$ number of kernels

Output : $w_{out} \times h_{out} \times d_{out}$

A convolutional layer is not only a convolution operation.
It consists of three stages:

- convolutions between input and kernel : still a linear operation
- non-linear activation function (detector), applied to each feature map
- pooling : another kernel that changes the output size ; it only affects width and height





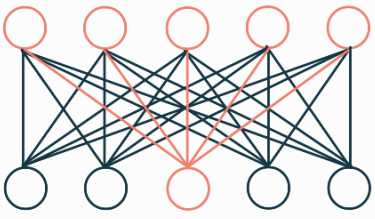
Pooling does not introduce trainable parameters
The depth of the output only depends on the depth of the kernel.
The width and the height depends on the pooling stage

Convolutional stage

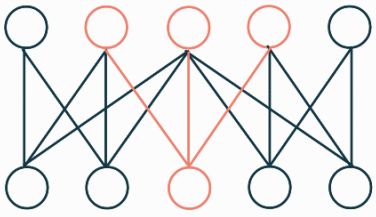
$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

The use of a kernel result in more efficiency in memory requirements and generalization accuracy. What happens when we use a Kernel?

① sparse connectivity:
outputs depend only on a few inputs; not all the nodes are connected with all the others



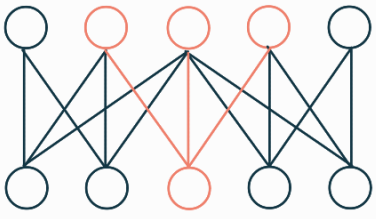
fully connected



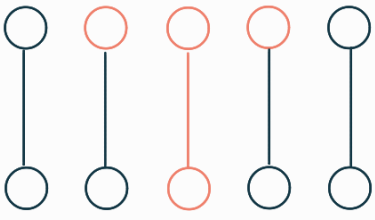
sparse connectivity: we could have a pattern in which a node will be connected only to e.g. 3 adjacent nodes on the next layer

We still have a good number of connections. Sparse connectivity highlights another important aspect: with sparse connectivity you can exploit locality

② parameter sharing:
aside from sparse connectivity we may want to impose that the value of some weights should be the same.
We will have k parameters instead of $m \times n$ ($k \ll m$)



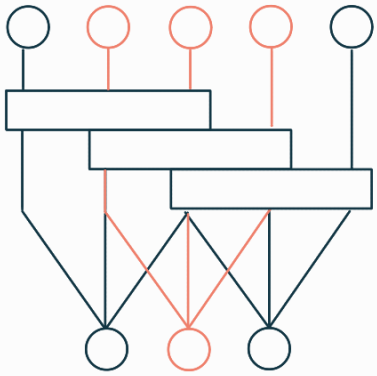
sparse connectivity



parameter sharing

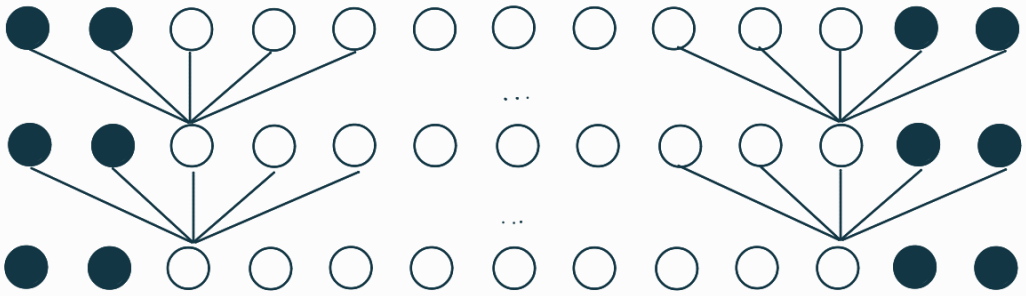
We keep the same parameters in the kernel

③ padding:



Directly applying the kernel will result on the shrinking of the input. If we want to preserve the spatial dimension of the input during the convolution process we need to add zeros around the border of the image. Not preserving spatial information in the image will lead to loss of information at the edges.

Output = Input - Kernel + 1



Padding size is usually $p = \lfloor \frac{w_k}{2} \rfloor$
Padding = how many fillers we need to use to fill the input in such a way that the output, when applied this kernel, will have the same size of the input

Detector stage

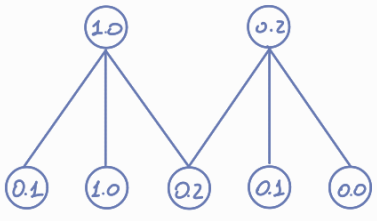
The detector stage consists on the application of an activation function to the result of the convolution stage. If we have insights on the problem we can choose an activation function, otherwise we can do experiments to find the best one or choose the ReLU.

Pooling stage

We need to reduce the output of the previous steps. We just define the size of the filter and a mathematical operation

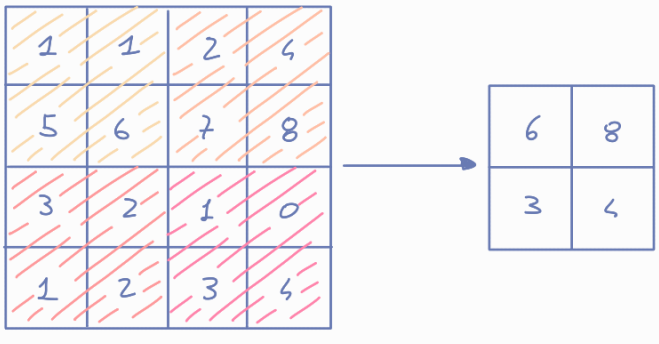
- max pooling: returns the maximum value in the rectangular region specified
- average pooling: returns the average value in the rectangular region specified

Example: max pooling with width 3 and stride 2



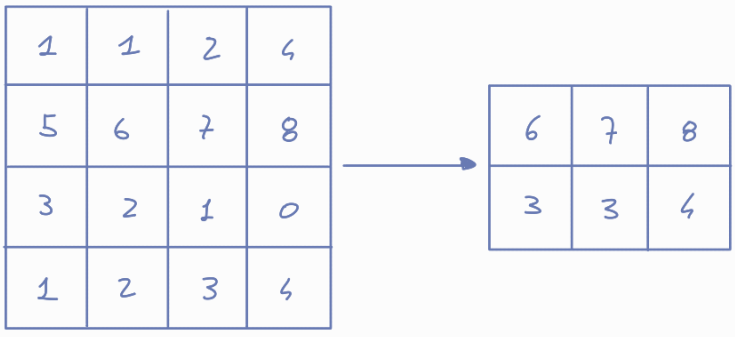
stride = how much we shift the filter (by default it is 1)

Example of max pooling of size 2x2 with stride 2



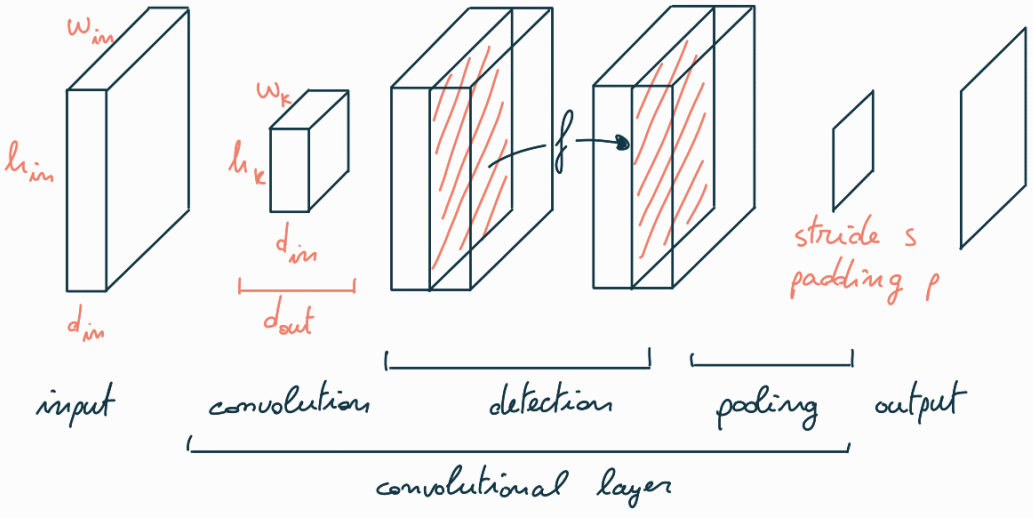
We can also use a different stride for vertical/horizontal movements. Useful if we want to shape the output

Example of max pooling of size 2×2 with stride $\overset{H}{1} \times \overset{V}{2}$



Pooling can be seen as a kind of subsampling (while the application of a kernel is a transformation of the input to another space).

Total number of parameters



Dimensions of output feature map are given by:

$$w_{out} = \frac{w_{in} - w_k + 2p}{s} + 1$$

$$h_{out} = \frac{h_{in} - h_k + 2p}{s} + 1$$

Number of trainable parameters:

$$|\theta| = \underbrace{w_k \cdot h_k \cdot d_{in} \cdot d_{out}}_{\text{kernel weights}} + \underbrace{d_{out}}_{\text{bias}}$$

one for each output feature map (d_{out})

this does not depend on the size of the input